The layout of the HDF5 file is roughly:

```
tstt/
 |
 +----elemtypes
 |
 +----nodes/
 |       |
 |       +---coordinates
 |       |
 |       +---tags/
 |
 +----elements/
 |       |
 |       +---"group"/
 |       |       |
 |       |       +---connectivity
 |       |       |
 |       |       +---tags/
 |       |
 |       +---"group"/
 |       |       |
 |       |       +---connectivity
 |       |       |
 |       |       +---tags/
...     ...
 |
 +----sets/
 |       |
 |       +----list
 |       |
 |       +----contents
 |       |
 |       +----parents
 |       |
 |       +----children
 |
 +----tags/
         |
         +---"name"/
         |       |
         |       +----type
         |       |
         |       +----id_list
         |       |
         |       +----values
         |
         +---"name"/
         |       |
         |       +----type
         |       |
         |       +----id_list
         |       |
         |       +----values
        ...
```

Introduction

MOAB's native file format is based on the HDF5 file format. The most common file extension used for such files is .h5m. A .h5m file can be identified by the top-level `tstt` group in the HDF5 file.

The API implemented by this library is a wrapper on top of the underlying HDF5 library. It provides the following features:

- Enforces and hides MOAB's expected file layout
- Provides a slightly higher-level API
- Provides some backwards compatibility for file layout changes

H5M File Layout

The H5M file format relies on the use of a unique entity ID space for all vertices, elements, and entity sets stored in the file. This ID space is defined by the application. IDs must be unique over all entity types (a vertex and an entity set may not have the same ID.) The IDs must be positive (non-zero) integer values. There are no other requirements imposed by the format on the ID space.

Elements, with the exception of polyhedra, are defined by a list of vertex IDs. Polyhedra are defined by a list of face IDs. Entity sets have a list of contained entity IDs, and lists of parent and child entity set IDs. The set contents may include any valid entity ID, including other sets. The parent and child lists are expected to contain only entity IDs corresponding to other entity sets. A zero entity ID may be used in some contexts (tag data with the mhdf_ENTITY_ID property) to indicate a 'null' value,

Element types are defined by the combination of a topology identifier (e.g. hexahedral topology) and the number of nodes in the element.

The `tstt` Group

All file data is stored in the `tstt` group in the HDF5 root group. The `tstt` group may have an optional scalar integer attribute named `max_id` . This attribute, if present, should contain the value of the largest entity ID used internally to the file. It can be used to verify that the code reading the file is using an integer type of sufficient size to accomodate the entity IDs.

The `tstt` group contains four sub-groups, a datatype object, and a data set object. The three sub-groups are: `nodes`, `elements`, `sets`, and `tags`. The data set is named `history` .

The `elemtypes` datatype is an enumeration of the elem topologies used in the file. The element topologies understood by MOAB are:

- `Edge`
- `Tri`
- `Quad`
- `Polygon`
- `Tet`

- Pyramid
- Prism
- Knife
- Hex
- Polyhedron

The history data set

The history data set is a list of variable-length strings with appliation-defined meaning.

The nodes Group

The nodes group contains a single data set and an optional subgroup. The tags subgroup is described in the section on dense tag storage.

The coordinates data set contains the coordinates of all vertices in the mesh. The data set should contain floating point values and have a dimensions n x d, where n is the number of vertices and d is the number of coordinate values for each vertex.

The coordinates data set must have an integer attribute named start_id . The vertices are then defined to have IDs beginning with this value and increasing sequentially in the order that they are defined in the coordinates table.

The elements Group

The elements group contains an application-defined number of subgroups. Each subgroup defines one or more mesh elements that have the same topology and length of connectivity (number of nodes for any topology other than Polyhedron.) The names of the subgroups are application defined. MOAB uses a combination of the element topology name and connectivity length (e.g. "Hex8".).

Each subgroup must have an attribute named element_type that contains one of the enumerated element topology values defined in the elemtypes datatype described in a previous section.

Each subgroup contains a single data set named connectivity and an optional subgroup named tags. The tags subgroup is described in the section on dense tag storage.

The connectivity data set is an n x m array of integer values. The data set contains one row for each of the n contained elements, where the connectivity of each element contains m IDs. For all element types supported by MOAB, with the exception of polyhedra, the element connectivity list is expected to contain only IDs corresponding to nodes.

Each element connectivity data set must have an integer attribute named start_id . The elements defined in the connectivity table are defined to have IDs beginning with this value and increasing sequentially in the order that they are defined in the table.

The `sets` Group

The `sets` group contains the definitions of any entity sets stored in the file. It contains 1 to 4 data sets and the optional `tags` subgroup. The `contents`, `parents`, and `children` data sets are one dimensional arrays containing the concatenation of of the corresponding lists for all of the sets represented in the file.

The `lists` data set is a `n x 4` table, having one row of four integer values for each set. The first three values for each set are the indices into the `contents`, `children`, and `parents` data sets, respectively, at which the *last* value for the set is stored. The contents, child, and parent lists for sets are stored in the corresponding data sets in the same order as the sets are listed in the `lists` data set, such that the index of the first value in one of those tables is one greater than the corresponding end index in the *previous* row of the table. The number of content entries, parents, or children for a given set can be calculated as the difference between the corresponding end index entry for the current set and the same entry in the previous row of the table. If the first set in the `lists` data set had no parent sets, then the corresponding index in the third column of the table would be `-1`. If it had one parent, the index would be `0`. If it had two parents, the index would be `1`, as the first parent would be stored at position 0 of the `parents` data set and the second at position 1.

The fourth column of the `lists` data set is a series of bit flags defining some properties of the sets. The four bit values currently defined are:

- 0x1 owner
- 0x2 unique
- 0x4 ordered
- 0x8 range compressed

The fourth (most significant) bit indicates that, in the `contents` data set, that the contents list for the corresponding set is stored using a single range compression. Rather than storing the IDs of the contained entities individually, each ID `i` is followed by a count `n` indicating that the set contains the contiguous range of IDs `[i, i+n-1]`.

The three least significant bits specify intended properties of the set and are unrelated to how the set data is stored in the file. These properties, described briefly from least significant bit to most significant are: contained entities should track set membership; the set should contain each entity only once (strict set); and that the order of the entries in the set should be preserved.

Similar to the `nodes/coordinates` and `elements/.../connectivity` data sets, the `lists` data set must have an integer attribute named `start_id`. IDs are assigned to to sets in the order that they occur in the `lists` table, beginning with the attribute value.

The `sets` group may contain a subgroup names `tags`. The `tags` subgroup is described in the section on dense tag storage.

The `tags` Group

The `tags` group contains a sub-group for each tag defined in the file. These sub-groups contain the definition of the tag and may contain some or all of the tag values associated with entities in the

file. However, it should be noted that tag values may also be stored in the "dense" format as described in the section on dense tag storage.

Each sub-group of the `tags` group contains the definition for a single tag. The name of each sub-group is the name of the corresponding tag. Non-printable characters, characters prohibited in group names in the HDF5 file format, and the backslash ('\') character are encoded in the name string by a backslash ('\') character followed by the ASCII value of the character expressed as a pair of hexadecimal digits. Thus the backslash character would be represented as `\5C` . Each tag group should also contain a comment which contains the unencoded tag name.

The tag sub-group may have any or all of the following four attributes: `default`, `global`, `is_handle`, and `variable_length`. The `default` attribute, if present, must contain a single tag value that is to be considered the 'default' value of the tag. The `global` attribute, if present, must contain a single tag value that is the value of the tag as set on the mesh instance (MOAB terminology) or root set (ITAPS terminology.) The presence of the `is_handle` attribute (the value, if any, is meaningless) indicates that the tag values are to be considered entity IDs. After reading the file, the reader should map any such tag values to whatever mechanism it uses to reference the corresponding entities read from the file. The presence of the `variable_length` attribute indicates that each tag value is a variable-length array. The reader should rely on the presence of this attribute rather than the presence of the `var_indices` data set discussed below because the file may contain the definition of a variable length tag without containing any values for that tag. In such a case, the `var_indices` data set will not be present.

Each tag sub-group will contain a committed type object named `type` . This type must be the type instance used by the `global` and `default` attributes discussed above and any tag value data sets. For fixed-length tag data, the tag types understood by MOAB are:

- opaque data
- a single floating point value
- a single integer value
- a bit field
- an array of floating point values
- an array of integer values

Any other data types will be treated as opaque data. For Variable-length tag data, MOAB expects the `type` object to be one of:

- opaque data
- a single floating point value
- a single integer value

For fixed-length tags, the tag sub-group may contain 'sparse' formatted tag data, which is comprised of two data sets: `id_list` and `values`. Both data sets must be 1-dimensional arrays of the same length. The `id_list` data set contains a list of entity IDs and the `values` data set contains a list of corresponding tag values. The data stored in the `values` table must be of type `type`. Fixed-length tag values may also be stored in the "dense" format as described in the following section. A mixture of both sparse- and dense-formatted tag values may be present for a single tag.

For variable-length tags, the tag values, if any, are always stored in the tag sub-group of the `tags` group and are represented by three one-dimensional data sets: `id_list`, `var_indices`, and `values`. Similar to the fixed-length sparse-formatted tag data, the `id_list` contains the IDs of the entities for which tag values are defined. The `values` data set contains the concatenation of the tag values for each of the entities referenced by ID in the `id_list` table, in the order that the entities are referenced in the `id_list` table. The `var_indices` table contains an index into the `values` data set for each entity in `id_list`. The index indicates the position of the *last* tag value for the entity in `values`. The index of the first value is one greater than the corresponding end index for the *entry* in `var_indices`. The number of tag values for a given entity can be calculated as the difference between the corresponding end index entry for the current entity and the previous value in the `var_indices` data set.

The `tags` Sub-Groups

Data for fixed-length tags may also be stored in the `tags` sub-group of the `nodes`, `sets`, and subgroups of the `elements` group. Values for given tag are stored in a data set within the `tags` sub-group that has the following properties:

- The name must be the same as that of the tag definition in the main

  `tags` group

- The type of the data set must be the committed type object stored

  as `/tstt/tags/<tagname>/type` .

- The data set must have the same length as the data set in the parent group with the `start_id` attribute.

  If dense-formatted data is specified for any entity in the group, then it must be specified for every entity in the group. The table is expected to contain one value for each entity in the corresponding primary definition table (`/tstt/nodes/coordinates` , `/tstt/elements/<name>/connectivity` , or `/tstt/sets/list)` , in the same order as the entities in that primary definition table.